

# TEHNICA DE PROGRAMARE "GREEDY"

## 1. Prezentare generală

Tehnica de programare Greedy este utilizată, de obicei, pentru rezolvarea problemelor de optimizare, adică a acelor probleme în care se cere determinarea unei submulțimi a unei mulțimi date pentru care se minimizează sau se maximizează valoarea unei funcții obiectiv. Formal, o problemă de optimizare poate fi enunțată astfel: "*Fie  $A$  o mulțime nevidă și  $f: \mathcal{P}(A) \rightarrow \mathbb{R}$  o funcție obiectiv asociată mulțimii  $A$ , unde prin  $\mathcal{P}(A)$  am notat mulțimea tuturor submulțimilor mulțimii  $A$ . Să se determine o submulțime  $S \subseteq A$  astfel încât valoarea funcției  $f$  să fie minimă/maximă pe  $S$  (i.e., pentru orice altă submulțime  $T \subseteq A, T \neq S$ , valoarea funcției obiectiv  $f$  va fi cel puțin /cel mult egală cu valoarea funcției obiectiv  $f$  pe submulțimea  $S$ ).*"

O problemă foarte simplă de optimizare este următoarea: "*Fie  $A$  o mulțime nevidă de numere întregi. Să se determine o submulțime  $S \subseteq A$  cu proprietatea că suma elementelor sale este maximă.*". Se observă faptul că funcția obiectiv nu este dată în formă matematică și nu se precizează explicit faptul că suma elementelor submulțimii  $S$  trebuie să fie maximă în raport cu suma oricărei alte submulțimi, acest lucru subînțelegându-se. Formal, problema poate fi enunțată astfel: "*Fie  $A \subseteq \mathbb{Z}, A \neq \emptyset$  și  $f: \mathcal{P}(A) \rightarrow \mathbb{R}, f(S) = \sum_{x \in S} x$ . Să se determine o submulțime  $S \subseteq A$  astfel încât valoarea funcției  $f$  să fie maximă pe  $S$ , i.e.  $\forall T \subseteq A, T \neq S \Rightarrow f(T) \leq f(S)$  sau, echivalent,  $\forall T \subseteq A, T \neq S \Rightarrow \sum_{x \in T} x \leq \sum_{x \in S} x$ .*"

Evident, orice problemă de acest tip poate fi rezolvată prin metoda forței-brute, astfel: se generează, pe rând, toate submulțimile  $S$  ale mulțimii  $A$  și pentru fiecare dintre ele se calculează  $f(S)$ , iar dacă valoarea obținută este mai mică/mai mare decât minimul/maximul obținut până în acel moment, atunci se actualizează minimul/maximul și se reține submulțimea  $S$ . Deși aceasta rezolvare este corectă, ea are o complexitate exponențială, respectiv  $\mathcal{O}(2^{|A|})$ !

Tehnica de programare Greedy încearcă să rezolve problemele de optimizare adăugând în submulțimea  $S$ , la fiecare pas, cel mai bun element disponibil din mulțimea  $A$  din punct de vedere al optimizării funcției obiectiv. Practic, metoda Greedy încearcă să găsească optimul global al funcției obiectiv combinând optimele sale locale. Totuși, prin combinarea unor optime locale nu se obține întotdeauna un optim global! De exemplu, să considerăm cel mai scurt drum posibil dintre București și Arad (un optim local), precum și cel mai scurt drum posibil dintre Arad și Ploiești (alt optim local). Combinând cele două optime locale nu vom obține un optim global, deoarece, evident, cel mai scurt drum de la București la Ploiești nu trece prin Arad! Din acest motiv, aplicare tehnicii de programare Greedy pentru rezolvarea unei probleme trebuie să fie însoțită de o demonstrație a corectitudinii (optimalității) criteriului de selecție pe care trebuie să-l îndeplinească un element al mulțimii  $A$  pentru a fi adăugat în soluția  $S$ .

De exemplu, să considerăm *problema plății unei sume folosind un număr minim de monede*. O rezolvare de tip Greedy a acestei probleme ar putea consta în utilizarea, la fiecare pas, a unui număr maxim de monede cu cea mai mare valoare admisibilă. Astfel, pentru monede cu valorile de 8\$, 7\$ și 5 \$, o sumă de 23\$ va fi plătită în următorul mod:  $23\$ = 2 \cdot 8\$ + 7\$ = 2 \cdot 8\$ + 1 \cdot 7\$$ , deci se vor utiliza 3 monede, ceea ce reprezintă o soluție optimă. Dacă vom considera monede cu valorile de 8\$, 7\$ și 1 \$, o sumă de 14\$ va fi plătită în următorul mod:  $14\$ = 1 \cdot 8\$ + 6\$ = 1 \cdot 8\$ + 6 \cdot 1\$$ , deci se vor utiliza 7 monede, ceea ce nu reprezintă o soluție optimă (i.e.,  $2 \cdot 7\$$ ). Mai mult, pentru monede cu 8\$, 7\$ și 5 \$, o sumă de 14\$ nu va putea fi plătită deloc:  $14\$ = 1 \cdot 8\$ + 6\$ = 1 \cdot 8\$ + 1 \cdot 5\$ + 1\$$ , deoarece restul rămas, de 1\$, nu mai poate fi plătit (evident, soluția optimă este  $2 \cdot 7\$$ ). În concluzie, acest algoritm de tip Greedy, numit *algoritmul casierului*, nu furnizează întotdeauna o soluție optimă pentru plata unei sume folosind un număr minim de monede. Totuși, pentru anumite valori ale monedelor, el poate furniza o soluție optimă pentru orice sumă dată (de exemplu, pentru monedele din Statele Unite ale Americii: <https://personal.utdallas.edu/~sxb027100/cs6363/coin.pdf>)

Revenind la problema determinării unei submulțimi  $S$  cu sumă maximă, observăm faptul că aceasta trebuie să conțină toate elementele pozitive din mulțimea  $A$ , deci criteriul de selecție este ca elementul curent din  $A$  să fie pozitiv (demonstrația optimalității este banală). Dacă mulțimea  $A$  nu conține niciun număr pozitiv, care va fi soluția problemei?

În anumite probleme, criteriul de selecție poate fi aplicat mai eficient dacă se realizează o prelucrare inițială a elementelor mulțimii  $A$  – de obicei, o sortare a lor. De exemplu, să considerăm următoarea problemă: "*Fie  $A$  o mulțime nevidă formată din  $n$  numere întregi. Să se determine o submulțime  $S \subseteq A$  având exact  $k$  elemente ( $k \leq n$ ) cu proprietatea că suma elementelor sale este maximă.*". Evident, submulțimea  $S$  trebuie să conțină cele mai mari  $k$  elemente ale mulțimii  $A$ , iar acestea pot fi selectate în două moduri:

- de  $k$  ori se selectează maximum din mulțimea  $A$  și se elimină (sau doar se marchează – important este ca, la fiecare pas, să nu mai luăm în considerare maximum determinat anterior), deci această soluție va avea complexitatea  $\mathcal{O}(kn)$ , care oscilează între  $\mathcal{O}(n)$  pentru valori ale lui  $k$  mult mai mici decât  $n$  și  $\mathcal{O}(n^2)$  pentru valori ale lui  $k$  apropiate de  $n$ ;
- sortăm crescător elementele mulțimii  $A$  și apoi selectăm ultimele  $k$  elemente, deci această soluție va avea complexitatea  $\mathcal{O}(k + n \log_2 n) \approx \mathcal{O}(n \log_2 n)$ , care nu depinde de valoarea  $k$ .

În plus, a doua variantă de implementare are avantajul unei implementări mai simple decât prima.

În concluzie, pentru o mulțime  $A$  cu  $n$  elemente, putem considera următoarea formă generală a unui algoritm de tip Greedy:

```

prelucrarea inițială a elementelor mulțimii A
S = []
for x in A:
    if elementul x verifică criteriul de selecție:
        S.append(x)
afișarea elementelor mulțimii S

```

Se observă faptul că, de obicei, un algoritm de acest tip are o complexitate relativ mică, de tipul  $\mathcal{O}(n \log_2 n)$ , dacă prin sortarea (prelucrarea) elementelor mulțimii  $A$  cu complexitatea  $\mathcal{O}(n \log_2 n)$  se poate ulterior testa criteriul de selecție în  $\mathcal{O}(1)$ . Dacă nu se realizează prelucrarea inițială a elementelor mulțimii  $A$ , atunci algoritmul (care trebuie puțin adaptat) va avea complexități de tipul  $\mathcal{O}(n)$  sau  $\mathcal{O}(n^2)$ , induse de complexitatea verificării criteriului de selecție. Evident, acestea nu sunt toate complexitățile posibile pentru un algoritm de tip Greedy, ci doar sunt cele mai des întâlnite!